IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES
(Attorney Docket № 13945US02)

In the Application of:

Uri Elzur et al.

Serial No.: 10/652,327

Filed: August 29, 2003

For: SYSTEM AND METHOD FOR NETWORK INTERFACING IN A MULTIPLE NETWORK ENVIRONMENT

Examiner: Hoang, Hieu T.

Group Art Unit: 2452

Confirmation No.: 1636

Electronically filed on April 6, 2010

RESPONSE TO NOTIFICATION OF NON-COMPLIANT APPEAL BRIEF

Mail Stop Appeal Brief – Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

This is a response to Notification of Non-Compliant Appeal Brief dated March 19, 2010 ("Non-Compliant Appeal Brief Notice"), with a reply period through April 9, 2010. The Appellant has attached herewith the Evidence Appendix (37CFR 41.37(c)(1)(ix), a copy of all non US patents or publication (items #3 and #6) entered by the Examiner, and relied upon by the Appellant in the Appeal Brief. The Appellant respectfully requests that the Non-Compliant Appeal Brief Notice be withdrawn.

EVIDENCE APPENDIX
(37 C.F.R. § 41.37(c)(1)(ix))

(1) United States Pat. No. 6,226,680 ("Boucher"), entered into record by the Examiner in the 8/13/2009 Final Office Action.

(2) United States Pub. App. No. 2002/0198934 ("Kistler"), entered into record by the Examiner in the 8/13/2009 Final Office Action.

(3) Microsoft Winsock Direct and Protocol Offload on SANs, 03/03/2001 ("Microsoft"), entered into record by the Examiner in the 8/13/2009 Final Office Action.

(4) United States Pub. App. No. 2002/0041566 ("Yang"), entered into record by the Examiner in the 8/13/2009 Final Office Action.

(5) United States Pub. App. No. 2003/0046330 ("Hayes"), entered into record by the Examiner in the 8/13/2009 Final Office Action.

(6) Callaghan (NFS over RDMA) ("Callaghan"), entered into record by the Examiner in the 8/13/2009 Final Office Action.

CONCLUSION

Based on at least the foregoing, the Applicant believes that all claims 1-31 are in condition for allowance. If the Examiner disagrees, the Applicant respectfully requests a telephone interview, and requests that the Examiner telephone the undersigned Patent Agent at (312) 775-8093.

The Commissioner is hereby authorized to charge any additional fees or credit any overpayment to the deposit account of McAndrews, Held & Malloy, Ltd., Account No. 13-0017.

A Notice of Allowability is courteously solicited.

Respectfully submitted,

Date: April 6, 2010

/ Frankie W. Wong /

Frankie W. Wong
Registration No. 61,832
Patent Agent for Applicant

McANDREWS, HELD & MALLOY, LTD.
500 WEST MADISON STREET, 34TH FLOOR
CHICAGO, ILLINOIS 60661
(312) 775-8093 (FWW)

Winsock Direct and Protocol Offload on SANs

Abstract: This paper provides information for system designers, high-speed interconnect hardware developers, and driver developers about protocol offload and how Microsoft® Winsock Direct (WSD) over System Area Networks (SAN) can radically reduce network protocol CPU and memory bottlenecks. This can increase system performance by freeing up CPU and memory bandwidth resources to be used by the application. The information in this paper applies for Windows 2000 Advanced Server (requires Service Pack 2), Windows 2000 Datacenter Server, Windows Server Appliance Kit (requires Service Pack 2), and all versions of the next release of Windows Server codename "Longhorn."

This paper summarizes traditional CPU and memory bottlenecks and provides a simplified version of the WSD protocol, called Trivial Transport Protocol (TTP), to help provide insight into solutions to the issues. Finally, it provides an analysis of the TTP protocol for how well it solves system bottlenecks. Note that while the TTP analysis applies directly to WSD, WSD is much richer in its message exchange than TTP to handle issues regarding cloning of sockets, small message optimizations, accumulated acknowledgements, and other issues.

Version 1.0— March 3, 2001

Contents

NFS over RDMA

Brent Callaghan

(brent.callaghan@sun.com)

Network bandwidth is growing by orders of magnitude. Yet conventional processing of NFS traffic over gigabit networks gobbles CPU. Using RDMA protocols, we expect NFS to make full and efficient use of gigabit networks.

January 28, 2002

1.0 Introduction

NAS file access protocols, like NFS, allow files to be shared among many servers running different operating systems. Yet this flexibility of NFS is tempered by its requirement for TCP/IP networks, which as yet, do not perform at the speed of SAN fibrechannel networks. Additionally, the protocol processing and data movement required by TCP/IP make heavy demands on the client and the server CPU.

The following sections describe a way to run file access protocols at high speed over fast storage networks without significant CPU overhead. You can enjoy the file sharing advantages of NFS with the performance of a SAN.

2.0 NFS Background

The NFS protocol was originally designed as a LAN protocol for file access over 10Mb Ethernet. Since the NFS protocol emerged in the mid-1980's, it has been used exclusively with, and strongly associated with TCP/IP Networking. The first NFS products ran over UDP, but in the last decade a growing number of implementations provide NFS over TCP connections. Work is under way within the Internet Engineering Task Force (IETF) to extend the NFS protocol to function well over TCP connections that span the Internet: NFS version 4. Like TCP/IP, NFS has a long history, implementations are very robust, the serious bugs were fixed long ago and NFS performance has increased significantly with improvements in host CPU and network bandwidth. Like TCP/IP, NFS implementations are available from all major vendors and yet it is highly interoperable due to annual testing at Connectathon events. It's a safe assumption that any NFS client will work with any Filer. NFS is well known as a tough protocol that is undeterred by transient network failures or Filer outages. Filers can be built from highly-available clusters and clients can easily switch between read-only, replica Filers. NFS provides a robustness that cannot easily be matched by block-based storage on SANs.

3.0 NFS in the Server Room

Today, a common NFS storage configuration is a pool of NFS filers that keep files for a large array of "stateless" application servers. Because the application servers do not have any dedicated storage and are not responsible for providing access to any storage, the failure of an application server doesn't block access to files. Since it keeps no files, a failed application server can be replaced easily. Application processing capacity can be increased simply by adding new servers.

The filers are located within a few meters of the application servers in a controlled, machine-room environment, and are connected to the application servers via fast or Gigabit Ethernet. The same machine rooms often host

FibreChannel connections from application hosts or filers to storage arrays.

4.0 Performance of Storage Networking

Typically, the FibreChannel connections move SCSI data significantly faster than Gigabit connections move NFS data, even though the media bandwidths are the same. Not only do the FibreChannel connections move data faster, but they use less host CPU than NFS running over TCP/IP. The FibreChannel and SCSI protocols are lightweight, short-haul protocols, easily implemented in I/O controllers. These controllers use Direct Memory Access (DMA) to move aligned data directly between the Fibrechannel network and host memory without burden to the host CPU. Credit is due to decades of I/O engineering that have produced architectures that offload protocol processing and data movement from the host CPU. File data moved via the NFS protocol has the potential to be just as fast, if it weren't for the movement of data through TCP/IP stacks that make heavier use of the host CPU and create additional data copies.

NFS has the potential to use similar high-speed networks and CPU offloading to accelerate file sharing to SAN speeds.

5.0 Direct Memory Access

Direct Memory Access (DMA) accelerates the movement of data between host memory and a network interface or I/O controller. Without DMA the device driver must use the host CPU to copy data between memory and the device buffers. DMA allows the device to "take over" the host memory bus and transfer the data itself, leaving the host CPU to work on less mundane work - like transaction processing. The device driver simply notifies the device where the data is (or where it is to go) then "kicks off" the DMA operation. Another benefit of DMA is that it allows a network adapter to use host main memory directly rather than depend entirely on local on-card buffering. This gives the device a lot of flexibility in handling very large data transfers in excess of its local buffering capacity.

6.0 Remote Direct Memory Access

Remote Direct Memory Access is an extension of DMA across an interconnect from one host to another. Rather than transfer data between host memory and a device buffer, RDMA moves data between memory on one host and memory on another host. As with DMA, the host processor(s) just "kick off" the transfer, while the details of moving the data from memory to memory via device buffers and network interconnect are handled completely by the hardware. Like DMA, RDMA is a hardware abstraction that is supported by a variety of application programming interfaces, hardware adapters, and network media. The most commonly implemented RDMA driver interface is VI (Virtual Interface) though more recent developments include the Direct Access Transport API (DAT) and RDMA support for Infiniband.

7.0 Doing RPC over RDMA

RDMA hardware gives us speedy and efficient movement of data from one host memory to another. If the of data represents an RPC call or reply message, then we have the basis for high-speed RPC transport. We are already familiar with the use of RPC over transports like UDP or TCP for NFS, or for memory-to-memory loopback operation between kernel and user address spaces for protocols like autofs. The RPC transport model already accommodates a variety of transports. It is straightforward to add RDMA as an additional transport without any need to modify the RPC protocol, or the application protocols that utilize it. The "plug-in" nature of RPC transports means that it is easy to provide a substantial acceleration for RPC-based protocols without change to the RPC applications, or their administration.

8.0 Use of queues and RDMA ops

The RDMA model for moving data is somewhat different to that of the "data stream" model of TCP connections or the "datagram" model of UDP sockets. With these more conventional models, data can be transmitted "in the blind". Streams or blocks of data can be transmitted to a network address, but the ultimate disposition of the data when it reaches that address is left completely up to the receiving system. The RDMA read and write operations require that the reader or writer have knowledge of the location and size of the destination buffer at both the source and the destination.

If you write a block of data from the memory of one host to another via RDMA you have to supply the virtual address of the destination buffer - and make sure that the buffer is big enough to accept all the data. Similarly, when reading data from a remote host, the virtual address from which the data is to be read must be known, as well as the location of a correctly sized local buffer. Both VI and Infiniband RDMA models provide message queues that allow short messages to be exchanged. These messages can be used to prompt RDMA read or write operations as well as providing source and destination virtual addresses and message size.

For example, if an RPC client needs to send a call message to a server, it might first send a short message to the server giving the virtual address and size of the call message. This is notification to the server that there is a call message waiting. On receiving this notification message, the server can allocate a buffer of the size needed, then initiate an RDMA read operation to transfer the RPC call message from the client to the server's buffer. When the RDMA read is complete, the server sends a short message to the client to notify it that the message has been received successfully.

A call message could also be sent to the server with an RDMA write operation. The client would send a message to the server requesting the address of a destination buffer big enough to hold the RPC call message. On receiving the server's response, the client initiates an RDMA write operation directly into the server's buffer. When the RDMA operation is complete, the client sends a short message to the server to notify it that the call message is transferred and waiting for attention.

Similar methods can be used to return RPC reply messages. There are lots of different ways that can be used to transfer RPC messages using a combination of RDMA operation and message queues. It is possible to optimize the process to minimize the number of messages exchanged and reduce latency as well as to adapt to differences in RDMA read and write speeds, to optimize bandwidth.

9.0 Additional services (NLM, ACL)

We have been looking at how RDMA can be used to accelerate RPC messages. Yet, we have referred only obliquely to the protocols that use RPC. Of course, the most important protocol in this context is the NFS protocol. However, the NFS protocol does not stand alone. File locking is handled by a separate Network Lock Manager protocol. Because this protocol is also RPC-based, it is also accelerated by the RDMA transport. Additionally, the NFS ACL protocol is used to provide control and viewing of Access Control Lists in filesystems that support them. Again, this protocol is RPC-based, and it too is accelerated by RDMA.

10.0 Protocol Offloading

The use of RDMA to accelerate NFS traffic can be used to improve performance for existing TCP/IP clients. Many customers use NFS filers to provide file services to TCP/IP clients. These clients cannot use RDMA acceleration directly because they are too far away from the Filers, or because they are low-cost workstations that cannot be outfitted economically with RDMA hardware. NFS filers for these clients must handle heavy TCP/IP processing loads when providing NFS services.

This TCP/IP processing load can be off-loaded to a box that moves NFS traffic between TCP/IP and RDMA networks. Incoming RPC calls can be extracted from their TCP/IP packets and sent on to the Filers via fast RDMA. RPC replies from the Filers are obtained via RDMA from Filer memory and returned to the client via TCP/IP. Because the Filer is no longer handling TCP/IP processing, its service latency is reduced and it can absorb a higher NFS load.

11.0 Benefits of NFS over RDMA

At first glance, the benefits of NFS over RDMA may appear to be just "faster NFS". Indeed, applications that already use NFS will benefit from the increased data bandwidth, reduced CPU overhead, and reduced latency.

But if NFS over RDMA performance matches that of "direct attach" or SAN-connected filesystems, then NFS is no longer a bottleneck, and we can appreciate the file sharing benefits of NFS more widely, even in applications that previously required "raw" disk access.

For example, a database provides high-volume, transaction-based access to large volumes of data. Typically, the database provides its own access methods, along with its own caching and I/O scheduling. It is normal for a database to have "raw" access to a disk partition and organize its own on-disk layout. In this role, a database fills a similar role to a filesystem, it maps data access semantics onto disk data blocks. The downside of having the database manage its own storage is that it burdens the system administrator with additional tools because database partitions are managed differently, e.g. UNIX system administration tools like "df" and "mount" do not work with database partitions. Databases do not benefit from filesystems features like automatic growth of file (table) space, or snapshots.

A database can benefit from filesystem features if tables are mapped onto files within a filesystem. With a properly tuned filesystem, a database can have good performance and benefit from more flexible filesystem administration. By far, the easiest filesystem storage for to install and manage is provided by NFS filers, so databases can have the best of both worlds: improved storage management through NFS filesystems, while maintaining performance.

12.0 Project Status

We have run a prototype implementation over an SCI (SCI (Scalable Coherent Interconnect) connection. The RPC layer in the client and server was modified to transport RPC messages as memory-to-memory operations. This experience confirmed that the RPC abstraction on which protocols like NFS are built, makes it quite easy to run RPC-based protocols over radically different transports.

We are further developing this RPC code to adapt readily to a variety of RDMA transports, such as VI, DAT (Direct Access Transport) and Infiniband®.

Introduction

WSD provides a true third-generation protocol offload, providing second-generation offload of the protocol stack and significant portions of the NDIS driver, as well as offloading the sockets layer and avoiding user-to-kernel space transitions. WSD radically reduces usage of precious CPU and memory bandwidth system resources. This reduction can result in improved application performance if the application is either CPU or memory bandwidth bound and has a significant network load. WSD has been architected to be SAN independent, so a wide variety of devices can be deployed with this protocol. Further, network wire speeds are increasing dramatically in the next few years, and WSD is well placed to enable these new technologies to avoid CPU and memory bottlenecks.

Typical applications that see substantial performance increase are database client-server applications, web servers at the front end of the data center, or application servers between the web server and the database server. The benefits of deploying a protocol offload SAN adapter that supports zero-copy algorithms are many, but can be summarized as making more CPUs and memory bandwidth available to applications.

For more information about Winsock Direct SAN Providers, see the Windows Driver Development Kit.

Increasing CPU and Memory Bandwidth Availability

In the data center, the capacity of many applications is limited by the availability of CPU or memory bandwidth. To increase the capacity of these datacenter applications, either faster CPU/memory architectures need to be deployed or protocol off-load mechanisms need to be examined. Off-load mechanisms work well for software modules that consume large portions of CPU or memory bandwidth and have clean interfaces. Networking protocol stacks, for example, are prime candidates for offloading.

Protocol stack overhead can directly affect the performance of server applications that are constrained by CPU or memory bandwidth. For example, database networking stacks can consume as much as one third of the total CPU utilization. If that consumption was reduced to 5%, then additional CPU resources would be available to the database applications. The database application would run 42% faster (0.95/0.66). Another way of thinking of it is that an 8-processor system is transformed into an 11.3 processor system by adding a System Area Networking (SAN) adapter that supports protocol offload.

Current first-generation network protocol offload algorithms provide TCP checksum offload and large send offload. Many vendors are pursuing a second-generation offload mechanism that is centered on offloading the transport stack (typically TCP) to the network adapter. Microsoft has focused on not only offloading the transport protocol stack, but also offloading or avoiding the work that occurs between the application and the protocol stack. Doing this provides superior offload characteristics compared to second-generation offload techniques. Note that this is not a research project—this capability is shipping in Windows 2000 Datacenter and Windows 2000 Advanced Server (Service Pack 2) today; existing application binaries work without modification.

WSD has been designed to be SAN independent. A SAN developer can write a SAN Provider to interface their network adapter to WSD. This is effectively a user-level device driver. The SAN Provider interface enables many varieties of SAN to be implemented underneath, including Compaq Servernet II, Giganet cLAN, Fibre Channel, and InfiniBand.

Characterization of Protocol Overhead

This section describes the common sources of protocol stack CPU and memory bandwidth utilization. Resource requirements for the networking stack will vary by workload. Only the common sources of the problem will be examined in this paper. Because priority varies by workload, no attempt will be made to prioritize the sources.

Memory Bandwidth Utilization

Typically, memory bandwidth is consumed by buffer copying (bCopies) because the received data from the network adapter did not arrive in the buffer that the application requires. The bCopy can occur within either the protocol stack or the application. If a bCopy occurs, it can occur on the transmit or receive of a packet, or it can occur on both transmit and receive.

Typically, when modern protocol stacks are used by well-written applications, no buffer copy is done on sends. On Windows, this is achieved by using asynchronous sends. Thus the area to focus on is the receive buffer copy. There is no known general-purpose algorithm for solving the receive copy problem for TCP/IP without the application being rewritten to accept buffers from the protocol stack instead of supplying buffers.

Note: This implies throwing out Winsock and coming up with a new API.

One algorithm that solves the problem in a limited way is called page-flipping. Page flipping requires that the application send only page-size data chunks and post receive buffers that are an integral number of pages in length. The basic idea is that the receive adapter has a single buffer pool to receive data into. If the adapter is smart enough to align the incoming data such that the Upper-Layer-Protocol (ULP) headers are separated from the payload, and the payload is an integral number of pages in length, then the operating system can manipulate the virtual to physical mapping of the receive buffer to "flip" the receive data into the application's virtual buffer. This mechanism is somewhat fragile—for example, if the transmitter or receiver sends a single message that does not obey the rules, or if the header is variable length — then page flipping will become misaligned. Because of the misalignment, a bCopy will be required from then on. Windows has chosen to not implement this algorithm because of its fragility and because commercial applications generally do not fit this profile.

It is common for current network applications to encounter a copy on receive. On common CPU/memory architectures, this consumes three times the memory bandwidth of the bCopy data rate. The following table describes the memory bandwidth requirements for common network fabrics if buffer copying is used.

Link	**Usable Link Rate one direction**	**Memory Bandwidth For bCopy**	**CPU's required at 200 MB/s bCopy rate**	**Total Memory Bandwidth with DMA and bCopy**	**WSD Availability**
Compaq Servernet II	125 MB/s	375 MB/s	1.9	500 MB/s	Now
Giganet CLAN	125 MB/s	375 MB/s	1.9	500 MB/s	Now
Fibre Channel	100 MB/s	300 MB/s	1.5	400 MB/s	Soon
Gig-Ethernet	125 MB/s	375 MB/s	1.9	500 MB/s	Soon
10 Gig-Ethernet	1250 MB/s	3750MB/s	18.8	5000 MB/s	N/A
InfiniBand 1x	250 MB/s	750 MB/s	1.25	1000 MB/s	2002
InfiniBand 4x	1000 MB /s	3000 MB/s	5	4000 MB/s	2002
InfiniBand 12x	4000 MB/s	12,000 MB/s	20	16,000 MB/s	2002

The previous table shows that for current network fabrics, memory architectures can barely keep up, and leave little extra bandwidth for the application. As an example, assume the CPU and memory system can maintain a 200 MB/s bCopy rate (this is aggressive for today's high performance systems). In this case, it is difficult for the system to have extra bandwidth for the application while also managing a single gigabit-per-second link. At 10 Gigabit Ethernet or InfiniBand 12x rates, the problem is essentially not solved—regardless of what the application is trying to do. Consuming 19-20 CPUs just to drive the network interface is not an effective solution for most applications.

CPU Utilization

The receive buffer copy is also often the largest source of CPU utilization. Other large sources include protocol overhead, kernel-to-user mode (and back) transitions, interrupt processing, and context switches because the receive or transmit completion event occurred in the wrong application thread.

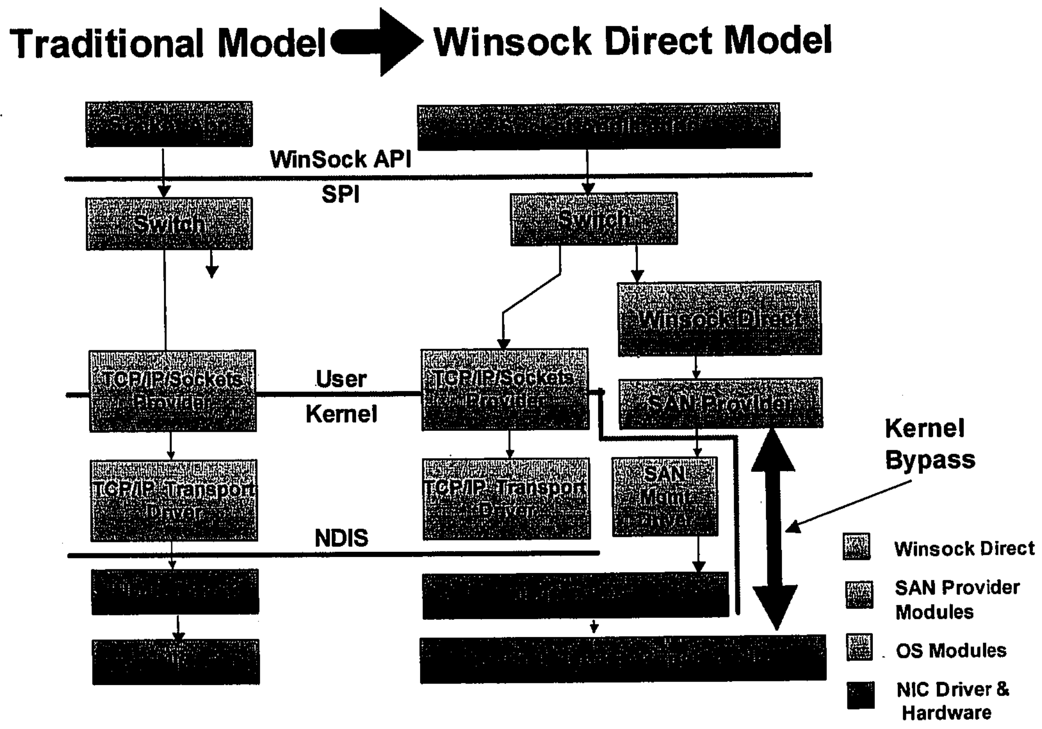
Transmit completion interrupts occur when the adapter has finished transmitting a buffer and has posted a transmit completion descriptor to the host. Receive completion interrupts occur when the adapter finishes receiving a packet and has posted a receive completion descriptor. Implementations vary from posting an interrupt per packet to providing some form of an interrupt moderation algorithm. This is possible when there are multiple completion descriptors in the queue.

CPU offload can also solve latency bottlenecks, which can slow down the application. This can directly effect distributed applications that have global synchronization points. Note that latency is not a significant problem for client-to-server database communications, however. Client-to-server database communications typically have many outstanding transactions, so the end-to-end latency of a single transfer is generally not in the critical path because operations are pipelined. If the database is distributed, however, then a global lock manager is in the critical path for server-to-server communication. This is highly latency sensitive, and can dominate performance of a parallel database server.

SAN and WSD Architectural Model

Figure 1 outlines the differences between the traditional networking model and Winsock Direct. The "switch" is the key to enabling WSD—it enables the application Winsock calls to be redirected to WSD rather than going down the conventional TCP/IP path. WSD then calls down to the SAN Provider to manage fabric and hardware specific issues. The SAN Provider is the equivalent to an NDIS driver, but in user space.

Figure 1: WSD and SAN Architectural Model



The Microsoft architectural model for SAN is to extend bus oriented semantics across the network. This includes the following characteristics:

- Reliable, in-order delivery with deterministic error models
- Two Transfer modes
  - Message semantics (also called Send semantics): packets are sent without specifying the destination buffer location.
  - Remote DMA (RDMA) semantics: packets are sent specifying the destination buffer location. RDMA semantics can include either an RDMA Write and/or an RDMA Read.

If Message semantics are used, the receiver must post buffers to the receive queue before the data arrives, or an error will result. If RDMA Write mode is used, the data sink must notify the data source how to access the receive buffer so that the data source can correctly specify the destination buffer location in the packet. If RDMA Read is used, the data source must notify the data sink how to access the buffer.

Typically, SAN hardware implements segmentation and reassembly (SAR) of buffers in hardware. This means that the I/O posted to the adapter can be larger than the maximum transfer unit (MTU) of the fabric. The SAN hardware will transparently segment the I/O into smaller packets and reassemble it at the destination.

Another typical feature of SANs is that they support sending and receiving data directly from or to a user application; the kernel is bypassed, which reduces overhead. An operating system that provides this kernel bypass capability can radically improve its packet-per-second throughput for small messages because little time is spent in the protocol stack and much of the overhead is due to the user-to-kernel space transition.

SANs usually provide a mechanism for interrupt moderation. By leveraging the kernel bypass capability, the SAN Provider can choose between a lightweight polling mechanism for interface activity (done in the user's context) or, under light load, fall back to the traditional interrupt-based schemes. For example, under heavy load conditions, typical network adapters can generate thousands of interrupts per second without interrupt moderation. Dynamically switching to polled mode during these conditions potentially means zero interrupts per second.

Because typical SANs provide kernel bypass capability, they typically have a lock for each TCP Port rather than a lock for the entire network adapter when transferring data. This substantially reduces lock contention and reduces CPU utilization.

Scenario: A Stream-Oriented Protocol

For purposes of illustration, a simple byte-stream oriented protocol will be defined, called Trivial Transport Protocol (TTP). TTP is a byte stream protocol that directly maps to WSD zero-copy behavior. Consequently, it could be mapped directly under the commonly used SOCK_STREAM sockets programming interface. To simplify the example, TTP requires the receiver to always post a buffer before the data transfer can begin and only uses RDMA Writes. WSD does not have this limitation and WSD enables both RDMA Reads and Writes, depending upon SAN capabilities. A control message called **RcvAvailable** is transmitted from the receiver to the transmitter to specify the receive buffer parameters. An RDMA Write is used by the transmitter to send the data directly to the application's buffer. This is followed by a send message called **WrComplete** that contains the TTP header for the RDMA Write data. The last message is required to allow the protocol headers to be demultiplexed into a separate buffer from the application buffer. See Figure 2 for TTP's buffering model, and Figure 3 for a ladder diagram showing the sequence of events.

Note: Because SAN hardware commonly implements hardware SAR, the size of the RDMA Write is the size of the buffer, not the MTU of the fabric. If the application transfers data using 64 KB buffers, for example, software needs to process 98% fewer packets. Implementing a SAR in hardware also reduces the number of transmit completion events or receive completion events when compared to a more conventional TCP implementation on Ethernet with a 1500 byte MTU.

Figure 2: TTP Buffering Model

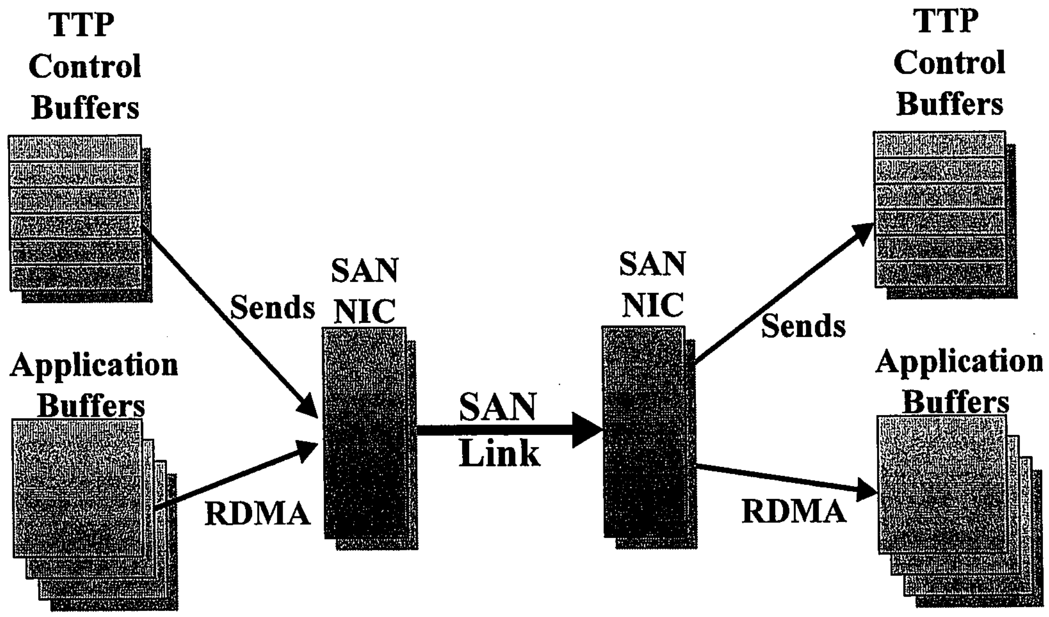
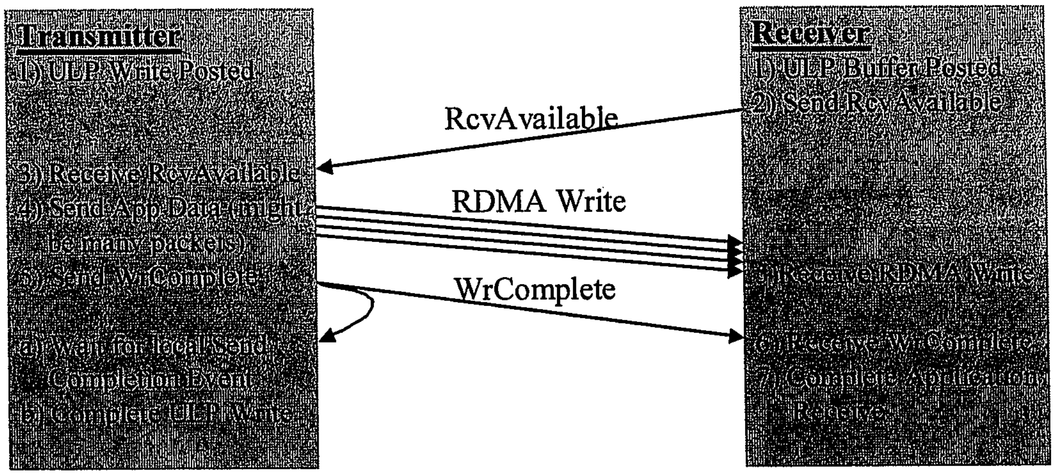


Figure 3: TTP Ladder Diagram



Because the control messages can be sent without incurring kernel transitions, user context switches, or copying of data—sending and receiving a message can have much lower overhead than typical protocol stacks. TTP's approach is that it is faster to generate a separate message for the header, allowing the demultiplex of the header from the ULP data, rather than to try to compress the header into the same packet as the ULP data. Another aspect of the reduced overhead is the repetition rate into the network pipeline has been greatly increased, so the packets-per-second achievable through WSD is significantly higher than conventional network fabrics. This is particularly appropriate for applications such as databases, which send a significant number of messages per second.

Analysis

By utilizing WSD hardware offload mechanisms, the major consumers of CPU and memory bandwidth can be addressed:

- Zero Copy Receives—RDMA Writes place the data directly into the application buffer
- Reduction in Protocol overhead
  - Retransmission timers and algorithms are in hardware
  - Reduced completion events

    --Segmentation and reassembly is in hardware

    --Completion events for transmit or receive of RDMA Write followed by **WrAvailable** can be combined. **RcvAvailable** transmit completion events can also be combined.
  - Reduced locking overhead—typical network adapters require a lock for each adapter. SAN typically requires a lock for each TCP Port, substantially reducing lock contention.
- Reduction in Receive and Transmit Interrupt Processing
  - Typical SAN implementations transition between a polled interface and an interrupt-based interface as load is reduced. Interrupts under high load can be effectively eliminated from the data transfer path under high load.
  - Fewer completion events also map to fewer interrupts

- Reduction in Kernel/User transitions

  SAN demultiplexes receive and transmit completions directly to the completion queue for that port, rather than to a central queue that is shared by all end points. This removes the requirement that the operating system be a central arbiter of completion events.

- User thread context switches

  By allowing the actual receive context to poll the completion queue directly, there are fewer context switches due to the wrong thread or process receiving the completion event